Testiranje integracije

U okviru ove lekcije govorimo o testiranju integracije ili integracionom testiranju. Ukoliko se podsetimo nivoa testiranja, dakle, prvi nivo testiranja je bilo testiranje jedinica, gde testiramo najmanje celine našeg programa. Drugi nivo je testiranje integracije, gde smo rekli da testiramo module ili pakete koda, i poslednji korak je testiranje celog sistema, tj. završene aplikacije. Sada smo, dakle, na drugom nivou, gde testiramo module i veze između njih.

Integraciono testiranje, kao što sam naziv govori, testira integracije. Pod pojmom integracije govorimo o vezama, interfejsima između komponenata programa kada se izvrši njihovo spajanje u jednu celinu. Kod integracionog testiranja u obzir se uzimaju svi aspekti, od operativnog sistema, fajl sistema, pa čak i samog hardvera, sve do pojedinačnih metoda, funkcija, klasa i interfejsa samog programa.

Da bismo bolje razumeli važnost ovog vida testa, sagledajmo još jednom proces razvoja aplikacije sa aspekta testova, tačnije sa test drive development tačke gledišta. Dakle, kada započnemo rad na programu, prve celine, metode ili cele klase, testiramo unit testom, da proverimo da li dobijamo očekivane rezultate.

Kada napravimo više klasa sa par funkcionalnosti programa, dolazimo do trenutka kada sve one treba da počnu da rade zajedno. Tu nastupa integraciono testiranje, gde proveravamo da li te različite celine rade zajedno, pre nego što završimo ceo program.

Na nivou malog projekta, gde radi samo jedan programer, integraciono testiranje se obavlja intiutivno. Dakle, odmah nakon što uradite jednu funkcionalnost, proverićete kako ona radi u odnosu na program, jer ne želite da se kasnije vraćate i ispravljate greške.

Ali, prava potreba za integracionim testiranjem se javlja kada radimo u okviru organizacije; stoga, u narednom delu lekcije pogledaćemo jednu realnu situaciju u okviru razvojnih timova.

Primer potrebe za testiranjem integracije

Zamislimo da radimo u IT organizaciji koja je dobila projekat razvoja web prodavnice. Nakon što se odviju incijalne faze projekta, o kojima smo govorili na samom početku ovoga kursa, dolazimo do podele posla u razvojnom timu.

Svaki od programera u timu dobija zadatke razvoja modula (funkcionalnosti) poput:

1. Registracija, validacija i login
2. Prikaz i unos proizvoda
3. Funkcionalnost korpe
4. Naplata
5. Isporuka i praćenje pošiljke

Kada je svaki modul dodeljen programerima, svako od njih započinje kodiranje pojedinih funkcionalnosti na svom računaru. Tokom procesa izrade debuguju, hvataju izuzetke i razvijaju svoj modul. Kada se razvoj modula završi, svako od programera sprovodi unit testiranje na svojim modulima; pronalaze probleme u svom kodu i rešavaju ih. Nakon ovog

procesa, svako od njih je završio svoj modul i dolazi trenutak u kojem se svi moduli spajaju u jednu celinu, kreiraju se interfejsi između klasa, i prvi put, moduli međusobno dele podatke. U trenutku kada se moduli spoje na jednom računaru, počinju da se pojavljuju problemi, gde aplikacija ne radi kako je očekivano. Pojavljuju se greške poput: kada se korisnik uloguje, ne vidi proizvode koju su pre toga bili u korpi, ili npr. na računu se ne dodaju troškovi posebnog načina isporuke itd. Dakle, greške koje nisu vezane za pojedinačni modul, već za komunikaciju između modula.

Upravo je sledeći korak u rukama projektnog menadžera. On određuje koje je testove integracije potrebno izvršiti da bi se programeri uverili da celokupan program radi prema zahtevima koji su dogovoreni još u fazi planiranja softvera.

Na osnovu ovoga možete videti koja je realna potreba i svrha integracionih testova, a to je da se otkriju problemi, otklone greške i da se osigura komunikacija između delova koda tako da se više različitih funkcionalnosti ponaša kao jedna celina.

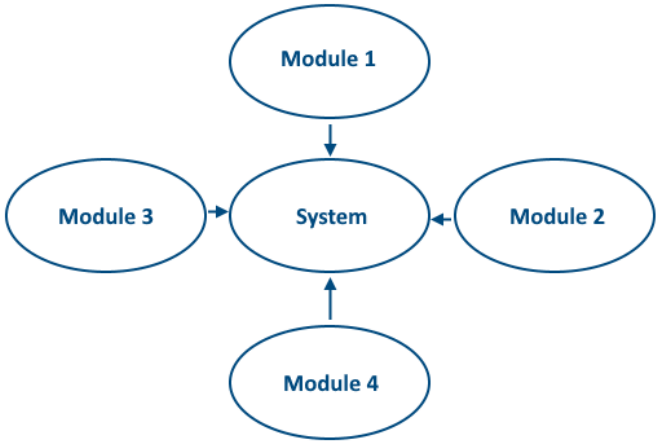
Tipovi integracije

Postoji više tipova integracije; izbor tipa testa koji koristimo zavisi od više faktora, od toga koliko novca ulažemo u testiranje, koliko kompleksne testove želimo, kolika je ozbiljnost grešaka aplikacije, ali možda najvažnije – od toga kako smo integraciju izvršili. U nastavku govorimo o najpopularnijim i najčešće korišćenim tipovima integracije.

Big bang integracija

Do pre nekoliko godina, ova integracija je bila uobičajeni model. Sastojala se od sledećih koraka:

1. Dizajn, kodiranje, testiranje, debagovanje svake klase
2. Kombinovanje klasa u veliki sistem
3. Testiranje i debagovanje čitavog sistema



Slika 6.1. Big bang integracija

Ovo je upravo vid integracije koji smo opisali u prethodnom primeru, gde sve module razvijamo pojedinačno i u jednom trenutku spajamo u sistem.

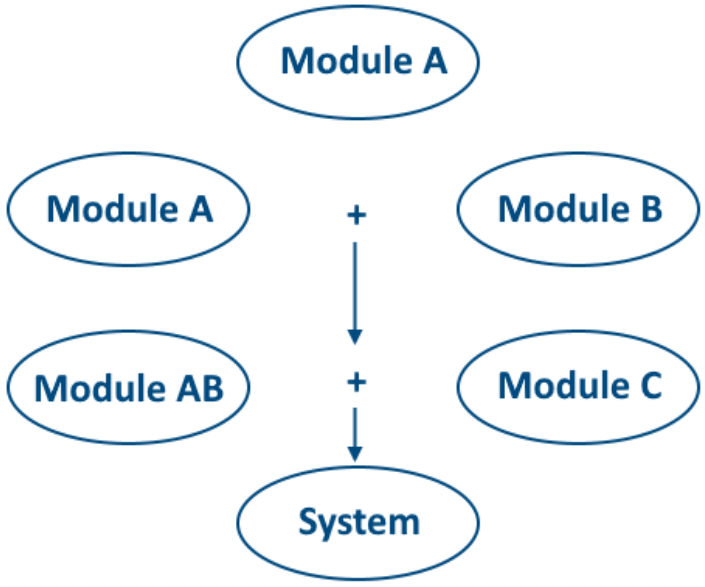
Problem sa korišćenjem ovog vida integracije je taj što se klase u sistemu spajaju zajedno prvi put, što neminovno dovodi do problema. Pošto postoji veliki broj klasa koje nikada pre nisu radile zajedno, uzrok može biti loše urađeno testiranje klase, greška u interfejsu između dve klase, ili greška prouzrokovana interakcijom između dve klase. Upravo iz navedenih razloga, integracija se teže obavlja, teže je napisati sve testove da bi se ona proverila i utvrdio tačan problem, ali ujedno, ovaj vid je odličan za male sisteme na kojima ne radi više od par programera. Tada se vrlo brzo sastavlja aplikacija i postoji manje problema, jer je manje ljudi radilo na njoj.

Inkrementalna integracija

U pristupu inkrementalne integracije, moduli različitih programera se integrišu jedan po jedan. Kako se poveže modul – testovi se sprovode odmah, i ovaj proces se ponavlja nakon svakog sledećeg modula.

Koraci ove integracije su:

1. Razvoj malog, funkcionalnog dela sistema. To može biti najmanji funkcionalni deo, najteži deo, ključni deo sistema i sl. Ovaj deo služi kao kostur, skelet koji će nositi ostale delove sistema.
2. Dizajnirati, kodirati, testirati i debagovati novu klasu.
3. Integrisati novu klasu sa kosturom.
4. Testirati i debagovati kombinaciju kostura i nove klase. Osigurati potpunu funkcionalnost ove kombinacije pre dodavanja nove klase. Ponavljati proces od tačke 2.



Slika 6.2. Inkrementalna integracija

Pristup inkrementalne integracije se odlikuje time da sve probleme nalazimo ranije, kada radimo sa manje koda u odnosu na završen proizvod. Ovo olakšava pronalaženje problema i ujedno pisanje testova za provere.

Preduslovi za pristupanje integracionom testiranju

U prvim lekcijama ovog kursa, govorili smo o tome da testiranje ne bi trebalo da sprovodi programer koji je napisao sam kod. Upravo ovo je vid testiranja koji autor koda ne bi trebalo da sprovodi, jer je tada programeru jako teško da uoči manu u kodu, a naročito kada je lično debagovao program, sprovodio unit test i pobrinuo se da sve radi po zahtevima.

U ovom slučaju se angažuje tester, koji prati korake poput koraka navedenih u nastavku:

1. Upoznavanje sa arhitekturom aplikacije
2. Prepoznavanje modula unutar programa
3. Razumevanje koja je svrha svakog zasebnog modula
4. Upoznavanje sa načinom kako se podaci kreću između modula
5. Podela aplikacije na celine koje odgovaraju za pisanje testa
6. Definisanje slučajeva testova (test case)
7. Prepoznavanje i kreiranje uslova ili prepostavki (assert)
8. Evidentiranje grešaka i razloga njihovog pojavljivanja

Primeri test slučajeva

Test slučajevi kod integracionih testova se primarno fokusiraju na interfejs između modula, linkove između njih, kao i razmenu podataka između njih. Razlog je to što su moduli već testirani unit testovima. Ovo stvara specifične stavke koje su obuhvaćene ovim testovima. Pogledajmo primer test slučaja integracionog testa za jedan program sa funkcijom društvene mreže.

Test slučajevi za korišćenje interfejsa:

- Testirati link između login strane i home page strane; konkretno, slučaj kada korisnik unese login podatke – trebalo bi ga automatski usmeriti na početnu stranu profila.
- Testirati link između početne strane i stranice koja vodi do profila korisnika. Klikom na link treba da se otvori stranica profila.

Test slučajevi za razmenu podataka između stranica:

- Ukoliko se na stranici za pregled zahteva potvrdi prijateljstvo, potrebno je da se zahtev ukloni sa profilne stranice, jer se i tamo prikazuju nepotvrđeni zahtevi.
- Ukoliko se odabere link na tasteru Comment, potrebno je da se prikaže prozor za unos teksta komentara.

Kao što možete videti, testovi integracije se uvek pišu sa određenom namenom – u zavisnosti od sajta, tipa aplikacije... Konkretno, ova četiri slučaja prikazuju samo situaciju kada radimo program sličan funkcionalnostima društvene mreže.

Odnos između unit testova i integracionih testova

S obzirom na to da je poenta integracionog testa da ispita odnose sastavljenih celina, ovih testova ima mnogo manje u odnosu na unit testove, koji ispituju na nižem nivou, ali važno je to da su oni u suštini jako slični. U kodu, često ne možete uočiti razliku da li je nešto unit test ili je to bio integracioni test. Pogledajmo primer.

Radno okruženje

```
def incrementX(x):
    return x + 1

def multiplyX(x):
    return 2 * x

def finalResult(x):
    return multiplyX(incrementX(x))

print (finalResult(2))
```

Objašnjenje:

Pogledajmo sledeće tri funkcije. Prva funkcija uvećava broj za jedan, druga funkcija koristi isti broj i množi ga sa dva, dok treća funkcija samo aktivira prethodne dve. Ove funkcije možemo zamisliti kao odvojene module – zašto? Svaka od ovih funkcija obavlja svoj posao, ali treća funkcija ih spaja u jednu celinu, gde postaju jedan program.

Prva i druga funkcija mogu biti unit testirane, jer svaka od njih radi svoju stvar i druge funkcije nemaju uticaj na nju, ali treća ne može biti unit testirana, jer se sav posao obavlja van nje. U okviru treće funkcije moramo izvršiti test integracije da se uverimo da je sve tačno i da dobijemo očekivan rezultat.

Ono što je olakšavajuće jeste da se integracioni testovi ne razlikuju u većoj meri od unit testova. U ovom slučaju, i dalje samo treba da uporedimo očekivani rezultat sa krajnjim rezultatom rada programa.

Stoga, test bi mogao da glasi:

```
def test_finalResult():
    expectedResult = 6
    result = finalResult(2)
    assert result == expectedResult
```

Kao što možete videti, ne postoji razlika u odnosu na unit test jedne funkcije, ali suštinski, obavili smo test integracije gde smo utvrdili da ceo program radi ono što je bilo očekivano od njega.

Izmeniti prethodni kod tako da se umesto multiplyX nalazi funkcija divideX koja deli prosleđeno x sa 2 a umesto funkcije incrementX funkcija decrementX koja smanjuje vrednost promenljive x za 1.

Važno je da ne pomislite da se ovo menja u odnosu na kompleksnost programa; iako je ovo jednostavan primer, isti princip će imati primer gde se promenila funkcionalnost u okviru ovih funkcija. Npr. da se preuzima vrednost korpe jednog shopa i dodaju fiksni porez na artikal i cena isporuke:

Radno okruženje

```
def basePrice(x):
    taxesFixed=5
    return x+taxesFixed

def shippingAdded(x):
    shippingAmazon = 12
    return x+shippingAmazon

def priceTotal(x):
    return basePrice(shippingAdded(x))


def test_finalResult():
    expectedResult = 316
    result = priceTotal(299)
    assert result == expectedResult
```

Objašnjenje:

Kao što vidite, princip je ostao isti; čak i da je svaka od ovih funkcija postala klasa sa par stotina linija koda, krajnji test se uopše ne bi promenio, jer i dalje poredimo ishode rada kao celine, što će biti samo par vrednosti koje treba uporediti da bismo potvrdili pravilan rad programa.

S obzirom na ove sličnosti, ostaje pitanje kada neki deo koda treba tretirati kao unit i nadalje ga testirati kao unit, a kada je potrebo da se neki deo našeg koda testira kao integracija. Testovi integracije mogu biti ekstremno jednostavni, kao što ste imali priliku da vidite u ovom primeru, ali mogu biti i veoma kompleksni jer će tražiti mnogo više test slučajeva. Ovo razlikovanje i samo pisanje testova postaju lakši ako posmatramo u odnosu na unit test.

Ukoliko funkcija ili klasa kombinuje/koristi dva ili više delova koda koji su ranije zahtevali unit testiranje, onda je potreban integracioni test. Ako funkcija uključuje neko novo ponašanje, koje pre toga nije testirano, potreban je unit test.

Kao što ste mogli da vidite, struktura samog integracionog testa se uopšte ne razlikuje od strukture unit testa. Dakle, i dalje koristimo assert; proveravamo neki uslov gde vršimo poređenje između toga šta se dogodilo sa onim šta je trebalo da se dogodi. Razlika je samo u tome što ćemo sada testirati rad više funkcija ili čitavih klasa na jednom mestu, umesto da testiramo jednu klasu ili njenu metodu. To će iziskivati da umesto npr. maksimalno desetak test slučajeva za jedan prosečan unit test, sada možete imati stotinu ili hiljadu uslova prilikom testiranja npr. programa koji je integrisan big bang integracijom. Sam princip nije teži, ali iziskuje više vremena i pažnje, jer sada razmišljamo o interakciji više modula našeg programa.

Pitanje

U kom pristupu se moduli svih programera integrišu jedan po jedan, tako da se testiranje sprovodi odmah i to ponavlja za svaki sledeći modul?

- Big bang integracija
- **Inkrementalna integracija**
- Unit integracija

Objašnjenje:

U pristupu inkrementalne integracije, moduli različitih programera se integrišu jedan po jedan. Kako se poveže modul – testovi se sprovode odmah, i ovaj proces se ponavlja nakon svakog sledećeg modula.

Rezime

- Integraciono testiranje se obavlja u svrhu testiranja veza/interfejsa između komponenata programa kada se izvrši njihovo spajanje u jednu celinu.
- Na nivou malog projekta, gde radi samo jedan programer, integraciono testiranje se obavlja intiutivno. Dakle, odmah nakon što uradimo jednu funkcionalnost, proveravamo kako ona radi u odnosu na ostatak programa, jer ne želimo da se kasnije vraćamo i ispravljamo greške.
- Postoji više tipova integracije i izbor tipa testa koji koristimo zavisi od više faktora – koliko novca ulažemo u testiranje, koliko kompleksne testove želimo, kolika je ozbiljnost grešaka aplikacije, i možda najvažnije – kako smo izvršili integraciju.
- Big bang integracija je osnovni pristup integraciji koda, koji odlikuje pojedinačni razvoj modula, koje u jednom trenutku spajamo u sistem. Problem sa korišćenjem ovog vida integracije je taj što se klase u sistemu tada spajaju prvi put, što neminovno dovodi do problema.
- U pristupu inkrementalne integracije, moduli različitih programera se integrišu jedan po jedan. Kako se poveže modul – testovi se sprovode odmah, i ovaj proces se ponavlja nakon svakog sledećeg modula.